Systems Engineering Process

Richard Shelton
Veridian
407-658-0044 x256
richard.shelton@veridian.com

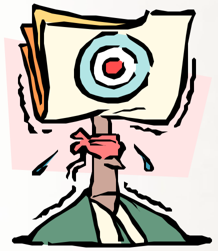
Agenda

- Process Overview
- Requirements Engineering/Analysis
- Design
- Development
- Integration
- Verification
- Gap Assessment
- Security
- Summary

Process Overview

Process, Process, Process

- The good engineers, designers, coders are ready and eager to jump in and build something, which is great!
- Lots of tools out there the help you sort, maintain, design, develop, test and such: DOORS, RTM, CORE, Rational Rose, Clearcase, Exceed, etc…

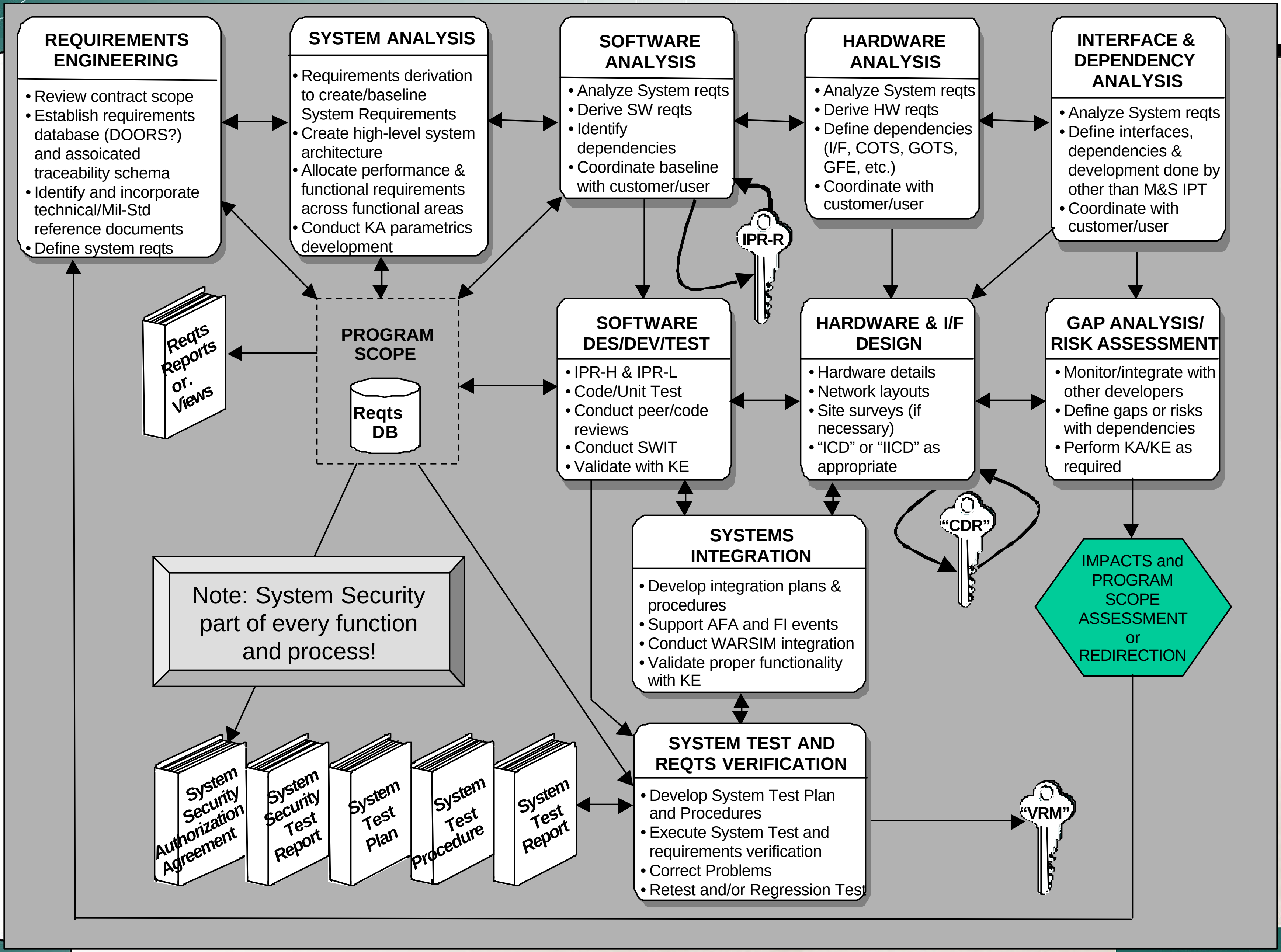


BUT, don't get ahead of yourself and not know where you are going

- Establish the plans and processes that define the life cycle of the program and know how to get there from here
- Lots of standards available to follow: ISO 9001, ISO 12207, CMM/SEI, Mil-Std-4998, Mil-Std-1521, etc…

VERIDIAN

Life Cycle Development Process Example



REQUIREMENTS ENGINEERING
- Review contract scope
- Establish requirements database (DOORS?) and assoicated traceability schema
- Identify and incorporate technical/Mil-Std reference documents
- Define system reqts

SYSTEM ANALYSIS
- Requirements derivation to create/baseline System Requirements
- Create high-level system architecture
- Allocate performance & functional requirements across functional areas
- Conduct KA parametrics development

SOFTWARE ANALYSIS
- Analyze System reqts
- Derive SW reqts
- Identify dependencies
- Coordinate baseline with customer/user

HARDWARE ANALYSIS
- Analyze System reqts
- Derive HW reqts
- Define dependencies (I/F, COTS, GOTS, GFE, etc.)
- Coordinate with customer/user

INTERFACE & DEPENDENCY ANALYSIS
- Analyze System reqts
- Define interfaces, dependencies & development done by other than M&S IPT
- Coordinate with customer/user

PROGRAM SCOPE

SOFTWARE DES/DEV/TEST
- IPR-H & IPR-L
- Code/Unit Test
- Conduct peer/code reviews
- Conduct SWIT
- Validate with KE

HARDWARE & I/F DESIGN
- Hardware details
- Network layouts
- Site surveys (if necessary)
- "ICD" or "IICD" as appropriate

GAP ANALYSIS/ RISK ASSESSMENT
- Monitor/integrate with other developers
- Define gaps or risks with dependencies
- Perform KA/KE as required

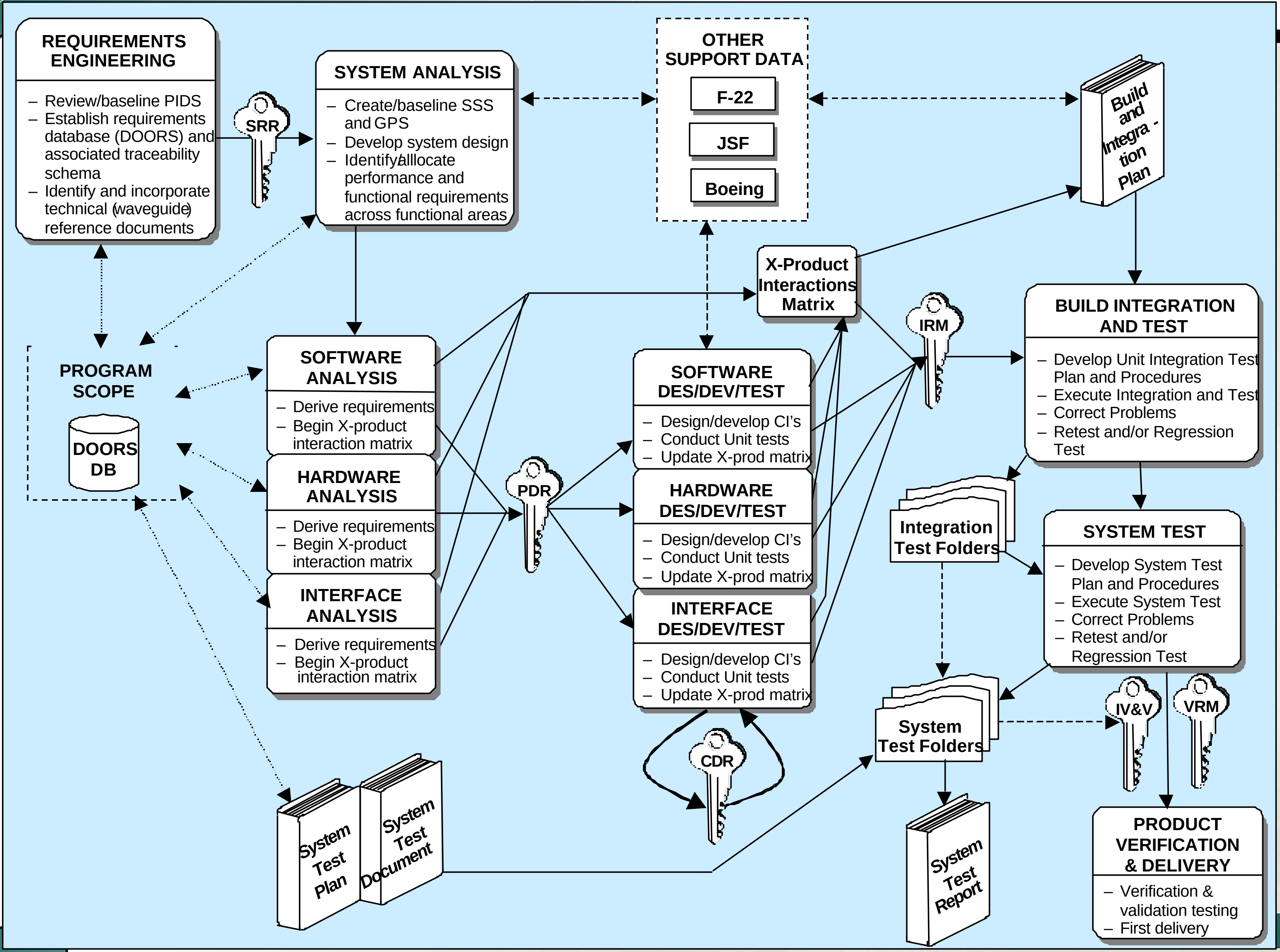
SYSTEMS INTEGRATION
- Develop integration plans & procedures
- Support AFA and FI events
- Conduct WARSIM integration
- Validate proper functionality with KE

SYSTEM TEST AND REQTS VERIFICATION
- Develop System Test Plan and Procedures
- Execute System Test and requirements verification
- Correct Problems
- Retest and/or Regression Test

IMPACTS and PROGRAM SCOPE ASSESSMENT or REDIRECTION

Note: System Security part of every function and process!

Another Life Cycle Development Process Example



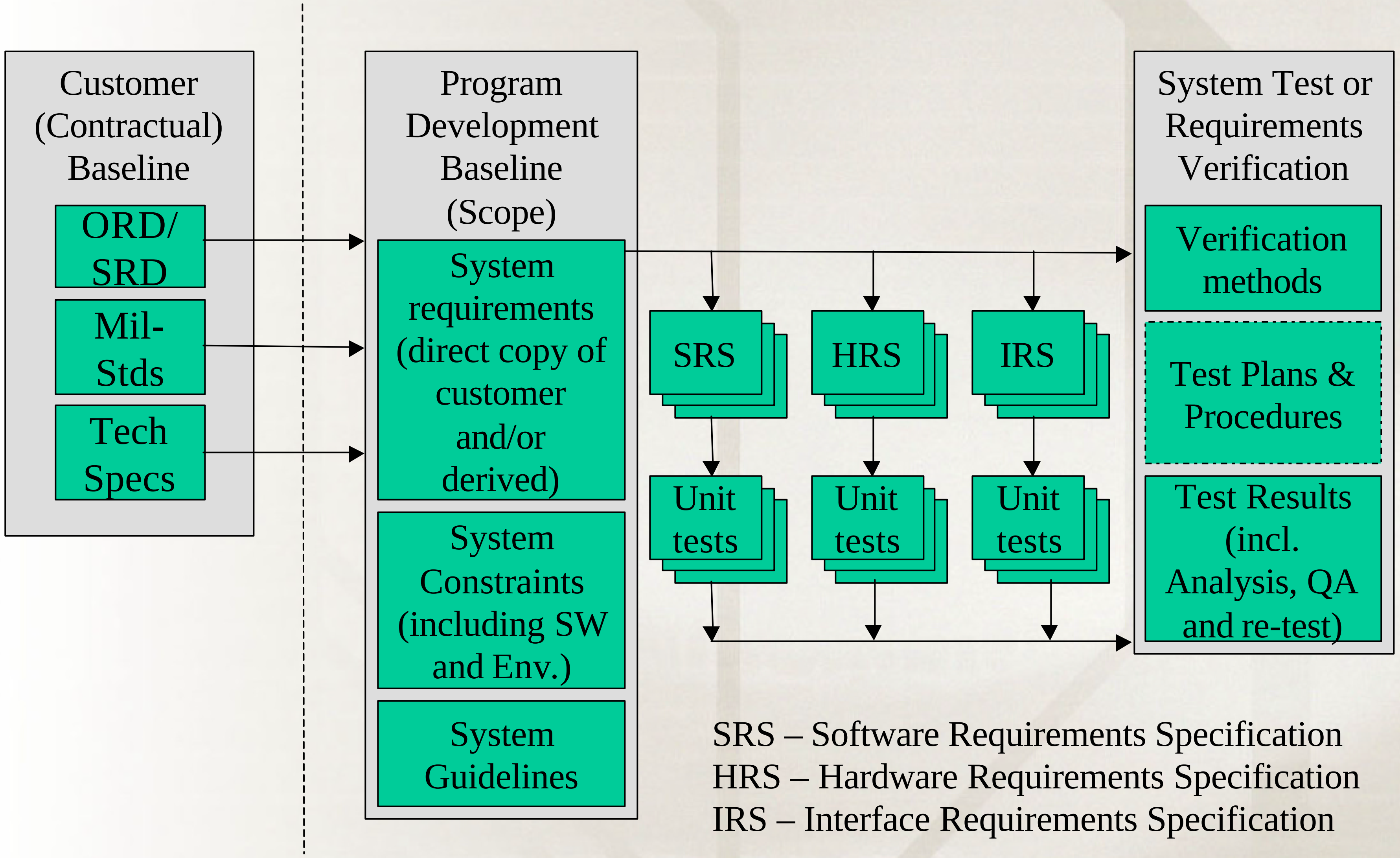
Requirements Engineering/Analysis

Most Critical Components

- Define the concepts from the user’s perspective for everyone to have the same vision
  - Everyone must see the vision to and talk the same language
  - Flow data from Concept Exploration of programs, from docs like:
    - Concept of Operations (CONOPS)
    - Conceptual Model of the User’s Space (CMUS)
- Provide a dedicated, comprehensive team to conduct a proper and sufficient requirements engineering
  - Include all functional areas (HW, SW, Safety, Security, ILS, Training, Interfaces, Test, Human Factors, etc.)
  - Understand time constraints and budget teams efforts accordingly
- Establish detailed schema for tracing requirements from users and customers to verification of final product

VERIDIAN

Generic Requirements Traceability Approach



Customer (Contractual) Baseline
ORD/ SRD
Mil-Stds
Tech Specs
Program Development Baseline (Scope)
System requirements (direct copy of customer and/or derived)
System Constraints (including SW and Env.)
System Guidelines
SRS
HRS
IRS
Unit tests
Unit tests
Unit tests
System Test or Requirements Verification
Verification methods
Test Plans & Procedures
Test Results (incl. Analysis, QA and re-test)

SRS – Software Requirements Specification
HRS – Hardware Requirements Specification
IRS – Interface Requirements Specification

Systematic requirements engineering is what Veridian excels in

VERIDIAN

Design

Some Design Basics

- Allocate your requirements across the functional areas of your program
  - General functional area examples: Hardware, Software, Interfaces (i.e. C4I), Safety, Security, Training
  - Specific functional area examples:
    - Hardware: avionics, communications, network
    - Software: sensor, platform, environment, database
- Let designers do their job, but provide process and overview direction
  - From a Systems Engineering standpoint, facilitate the process, helping define the "goes into" and "goes out of" (entry and exit criteria for this phase)
  - Let (make) the software and hardware designers design and derive the answers (let them share or take ownership of the problems)

VERIDIAN

Development

Development Basics

- Again, the System Engineer facilitates the process, helping define the entry and exit criteria, but let the developers develop
- Scope analysis and peer reviews are important
  - Developers take understandable pride in their efforts, and often want to make it the best possible product they can, but when is it too much?
    - Review what they are creating and make sure it meets requirements, but don’t lose scope control
    - Excess functionality takes time to develop that may be needed later in the program schedule
  - Use peer reviews as an integrated effort to give everyone a common understanding of the product

VERIDIAN

Integration

Start Small and Simple

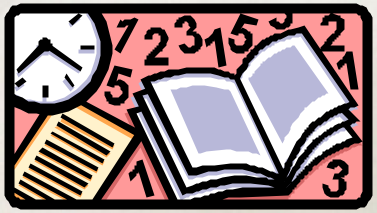
- Many programs take much longer than planned for integration
  - Trying to do a complete integration at once
  - Avoiding perceived “unnecessary” cost of testing “more than once”
- Integrate small pieces at a time by testing single functional threads at a time until you’re comfortable with interfaces that cross domain boundaries
- Let the developers and users provide insights into validity of behaviors and results
- Lean on the experience of component testers who know how the interfaces are supposed to work

VERIDIAN

Verification

Re-use and Traceability Analysis

- Verification proves to the user that you met the systems requirements he agreed to at the beginning of the program
- Remember that by this point, you've already tested the functionality several times through unit testing, integration testing, and some system testing
  - Use traceability and analysis to fold lower level testing, including test plans and procedures, up into the system verification
  - Your system requirements should trace down into each subcomponent and back up into an integrated system (reminder, build it into your requirements schema in the beginning)



Gap Assessment

Find the Holes Sooner than Later

- As you go along, incorporate periodic reviews of your requirements versus designs and products
  - Determine if all the requirements were properly "flowed down" and are being satisfied
  - People forget to go back and reread the CONOPS and requirements to remind them (and focus them) on the scope of their efforts
- Tracing requirements from top to bottom, and then back to top, is very complex
  - Double-check traces before blaming the designers/developers of missing something
  - Many times the design is there, but the traces are not

VERIDIAN

Security

Always Keep It in the Forefront

- Train your folks, multiple times if necessary, to make sure security is leading their designs and products
  - Putting in guards, firewalls, gateways or work-arounds later to correct poor security can be very, very expensive
  - Make sure your entire team understands the security vision and approach
- Know the requirements (NISPOM, DCID, etc.) and ways they can be met
  - Understand tools in industry that makes your security job and designs easier
  - It's always changing and getting better

VERIDIAN

Summary

Main Points

- Take the time to do the requirements engineering right the first time
- Many tools available to help take you from requirements to design, but remember basic principles:
  - Maintain focus and scope, don't burst your requirements bubble with "bells and whistles" or "requirements creep"
  - Get and maintain a common vision that everyone understands and works towards
  - Don't get caught in the weeds, let designers and developers do their job, but help them stay on track

VERIDIAN

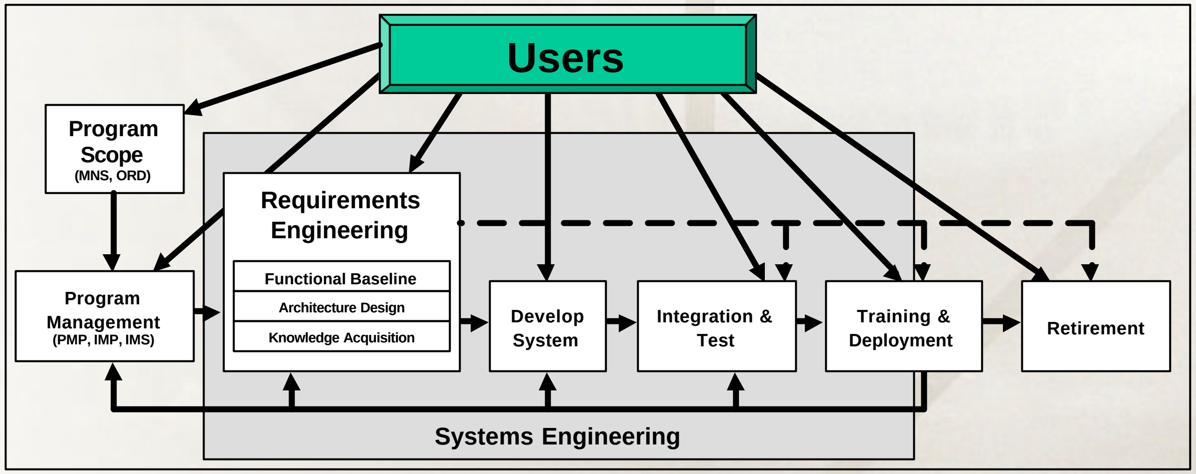
A Few Lessons Learned

- Keep the users involved in every step
- Tracing must be precise and complete
  - Else FRT (Forward Requirements Trace) and BRT (Backward Requirements Trace) will be useless
  - Any System requirement not traced downward (or properly “stubbed”) will be considered not satisfied and a development “hole”
  - Any Derived requirement not traced upward will be considered out of scope and not appropriate for development baseline
  - PLEASE don’t use internal links; structure the document so these will not be necessary (see Vern for details and help)
- Actual printed documentation (System Spec, SRS, HRS, etc.) will be outdated references and should only be printed and understood as being “dated” view of the development baseline
  - The current development baseline will only be in the configuration managed DOORS database
  - Everyone can view most current data immediately via tool, avoiding having to check if the paper copy is current, or what’s changed

VERIDIAN

An SBA Modeling and Simulation Perspective

- Simulation Based Acquisition (SBA) feeds and draws from the Systems Engineering Development Life Cycle
  - To help scope the program
  - Uses Modeling and Simulation to help
    - Bound and define the scope
    - Performance and effectivity of the functions and products before or as part of deriving the software, hardware, interface, safety and security requirements

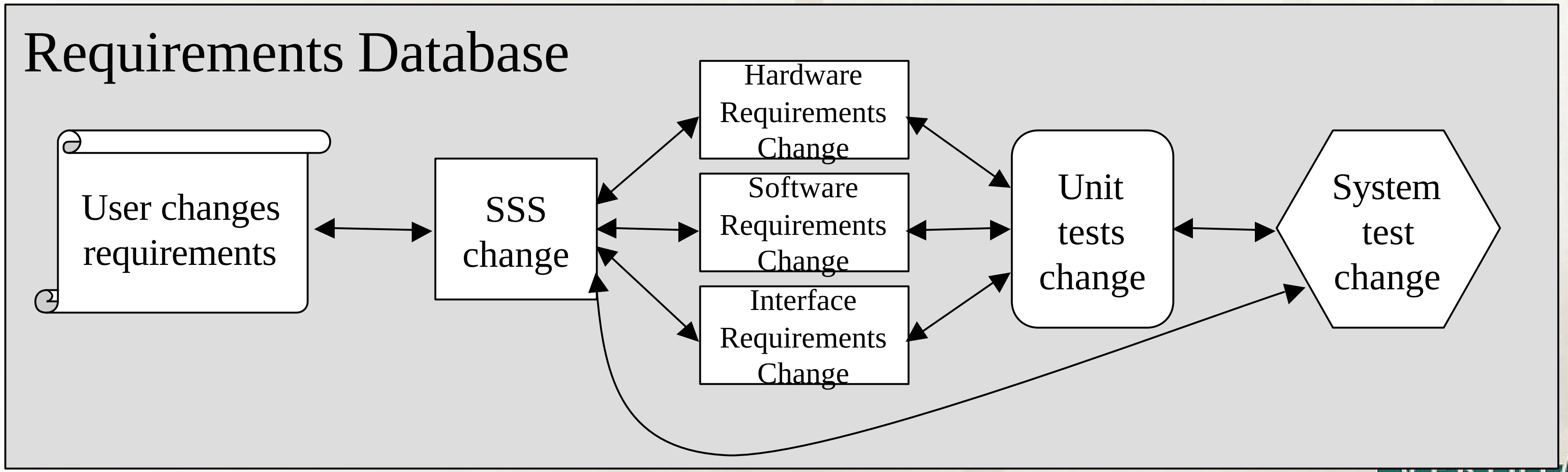


Back-up

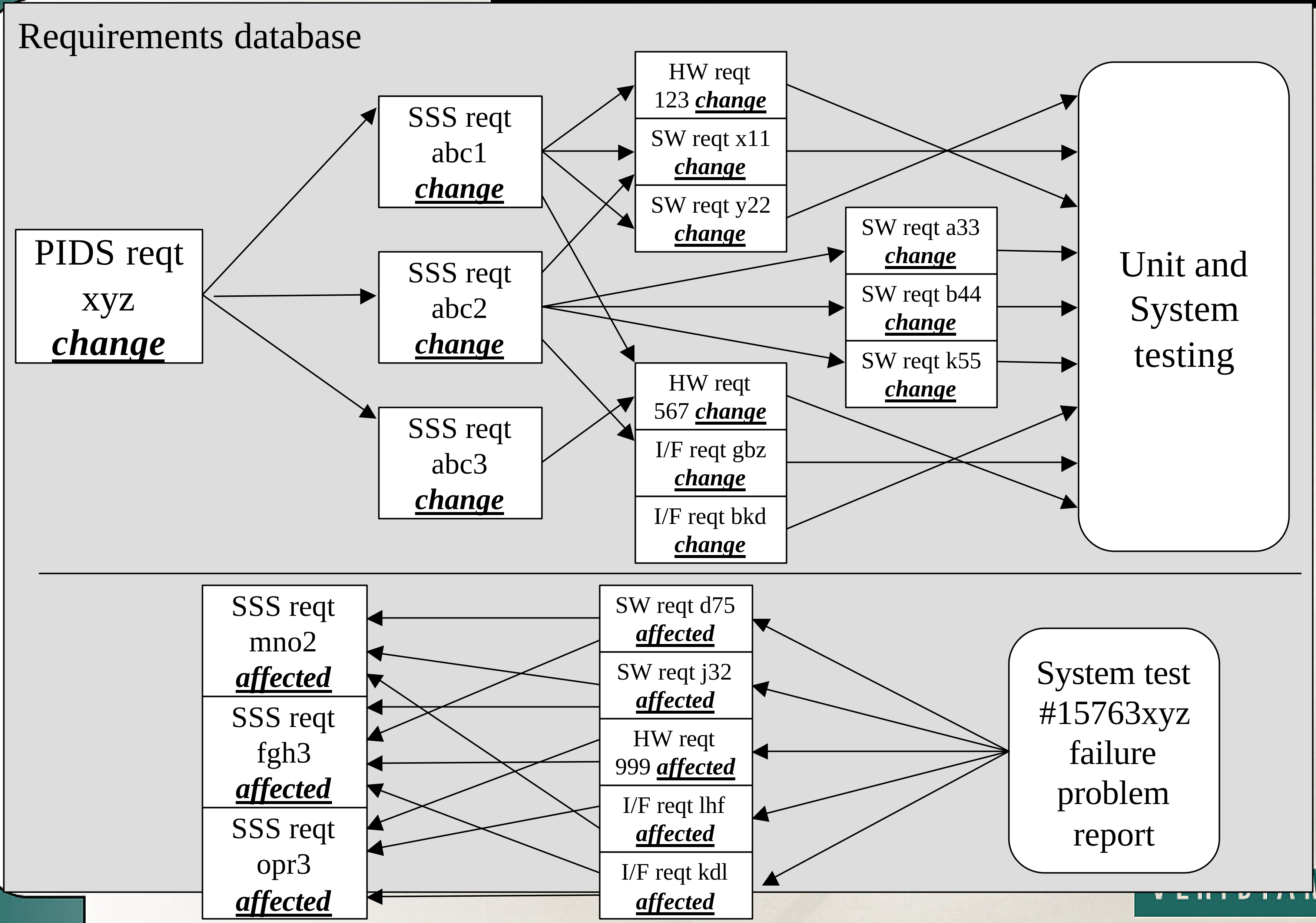
Traceability (Backup)

Effective Database Application/Traceability is Critical

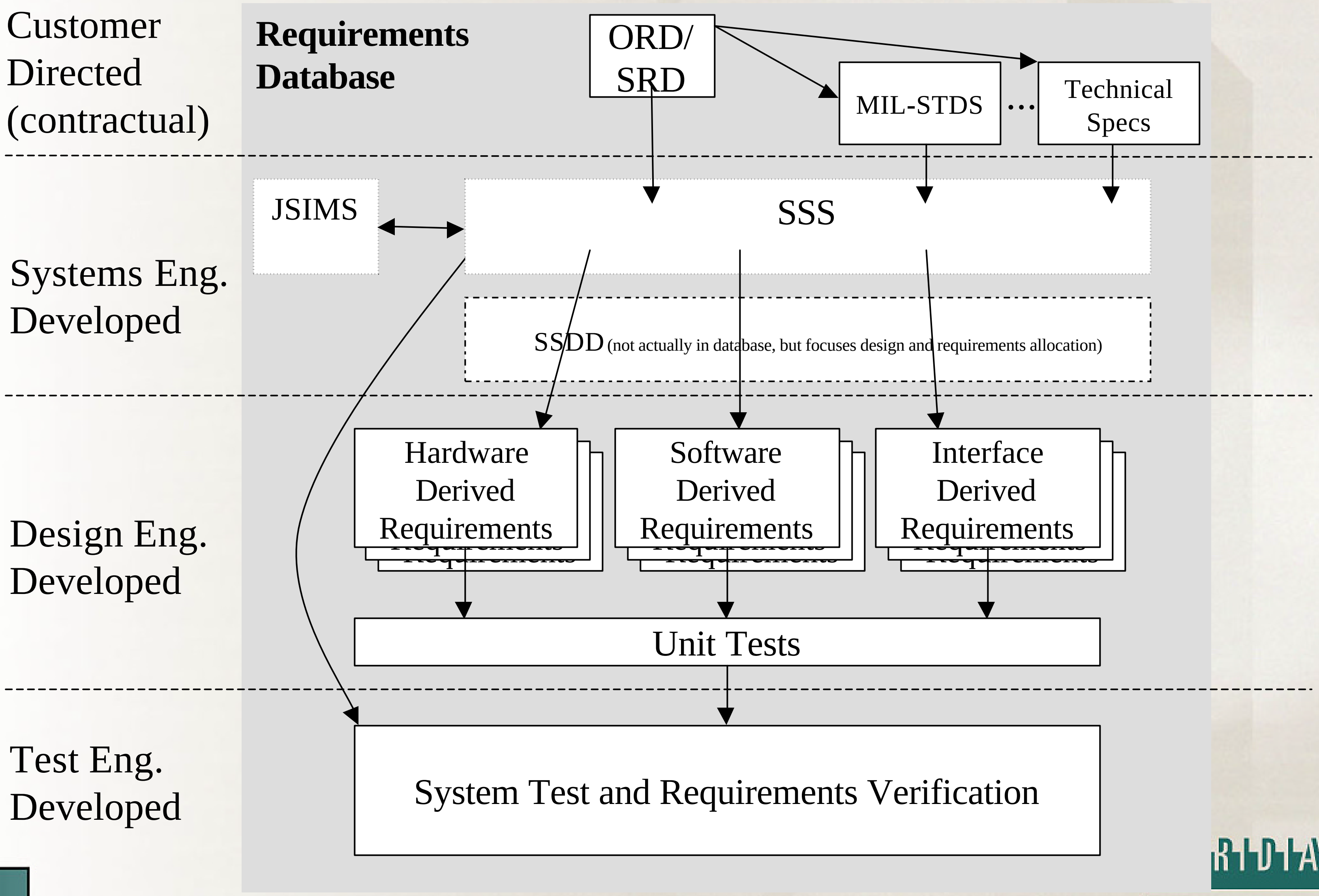
- Effective database application and traceability is critical to support:
  - Efficient configuration management and change control (database control of access and distribution)
  - Everyone (with proper access) can see current documentation and any updates or change history
  - Quick analysis or definition of change impacts (forward for requirements changes and reverse for design and test changes)
  - Support requirements verification and validation efforts
  - Easier SEI Level 3 compliance


Requirements Database
User changes requirements
SSS change
Hardware Requirements Change
Software Requirements Change
Interface Requirements Change
Unit tests change
System test change

Forward and Backwards Tracing


Requirements database
PIDS reqt xyz ***change***
SSS reqt abc1 ***change***
SSS reqt abc2 ***change***
SSS reqt abc3 ***change***
HW reqt 123 ***change***
SW reqt x11 ***change***
SW reqt y22 ***change***
SW reqt a33 ***change***
SW reqt b44 ***change***
SW reqt k55 ***change***
HW reqt 567 ***change***
I/F reqt gbz ***change***
I/F reqt bkd ***change***
Unit and System testing
SSS reqt mno2 ***affected***
SSS reqt fgh3 ***affected***
SSS reqt opr3 ***affected***
SW reqt d75 ***affected***
SW reqt j32 ***affected***
HW reqt 999 ***affected***
I/F reqt lhf ***affected***
I/F reqt kdl ***affected***
System test #15763xyz failure problem report

Example Database/Traceability Schema



Customer Directed (contractual)
Systems Eng. Developed
Design Eng. Developed
Test Eng. Developed
Requirements Database
ORD/ SRD
MIL-STDS
…
Technical Specs
JSIMS
SSS
SSDD (not actually in database, but focuses design and requirements allocation)
Hardware Derived Requirements
Software Derived Requirements
Interface Derived Requirements
Unit Tests
System Test and Requirements Verification

Requirements Development Guidelines

- Requirements tell you “what”, not “how”. Do not constrain designers by requiring certain implementations, hardware, etc.
- Derived requirements can have multiple levels, with the lowest level defining a single, testable “function”
- Write positive requirements, avoid putting “shall not…”, such as “…CSCI/HWCI shall not send …”
- Double-check to make sure requirements are testable and supportable
  - Be specific about “what” you require, and avoid open ended statements containing “may be”, “to include” or “might consist of”
  - If it’s a performance related requirement, then bound the statement (throughput required, word-size, cooling capacity, etc.) so it can be tested
  - Do not define your test method in the requirement, such as “this requirement may be verified by analysis (SSDD 4.1.0.0.2.1, 00-00040); leave that for the test folks to define during verification method definition
  - Do not reference other documents in total, as that may require testers to verify performance against the entire document, but rather be precise in what you reference; note: it’s best not to reference any external document to avoid constant updates or configuration management problems
  - Be sure technology and program resources support the requirements (i.e. don’t require hardwire or systems not available or unaffordable); besides, don’t write constraining design or implementation requirements, focus on functionality

VERIDIAN